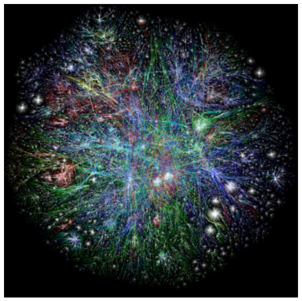
CS 225

Data Structures

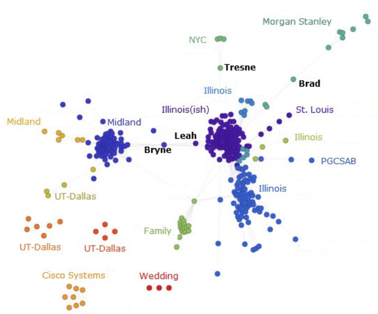
November 12 – Graphs

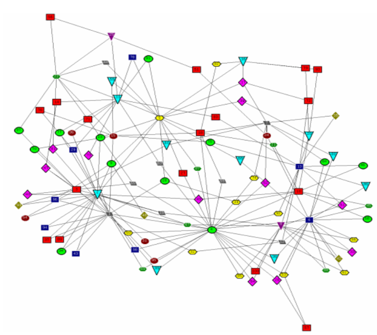
Wade Fagen-Ulmschneider

Graphs



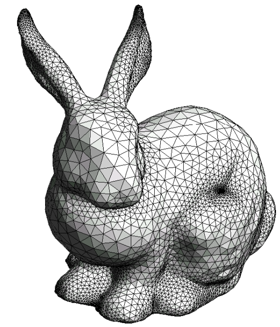
HAMLET   TROILUS AND CRESSIDA

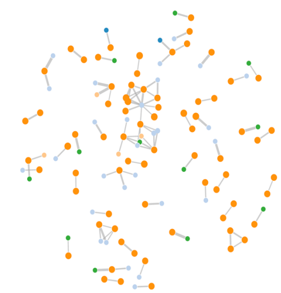


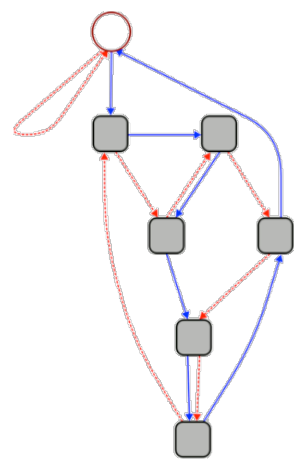


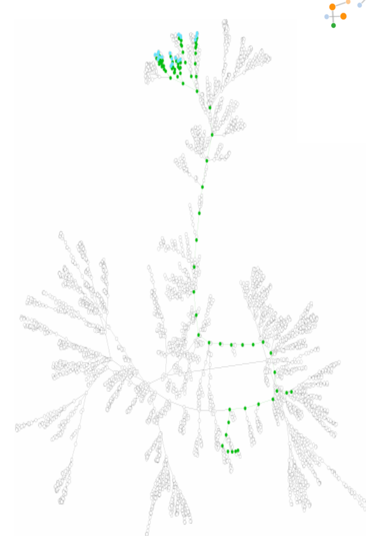
To study all of these structures:

1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms







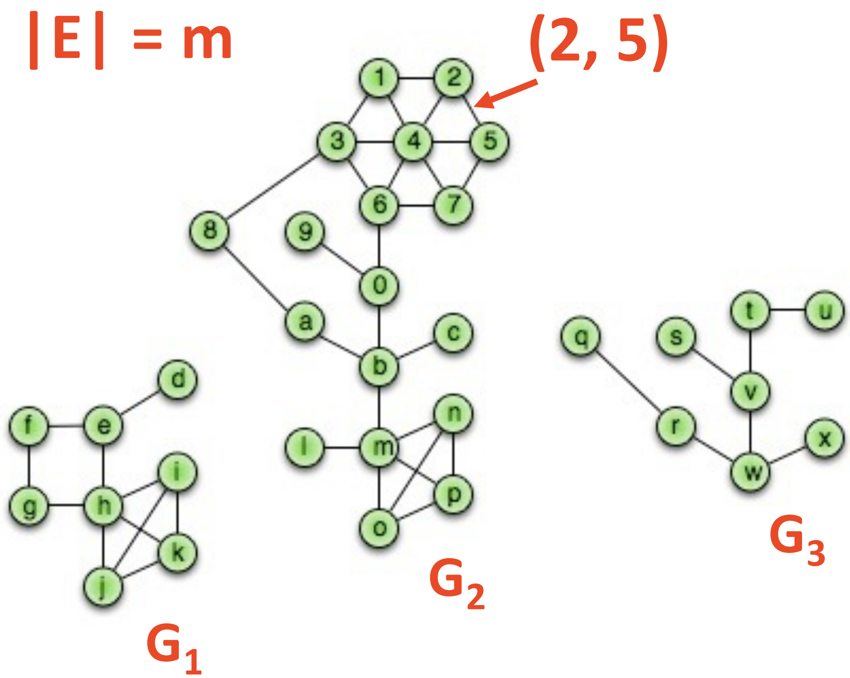


Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Incident Edges:
$I(v) = \{ (x, v) \text{ in } E \}$

Degree(v): $|I|$

Adjacent Vertices:
$A(v) = \{ x : (x, v) \text{ in } E \}$

Path(G_2): Sequence of vertices connected by edges

Cycle(G_1): Path with a common begin and end vertex.

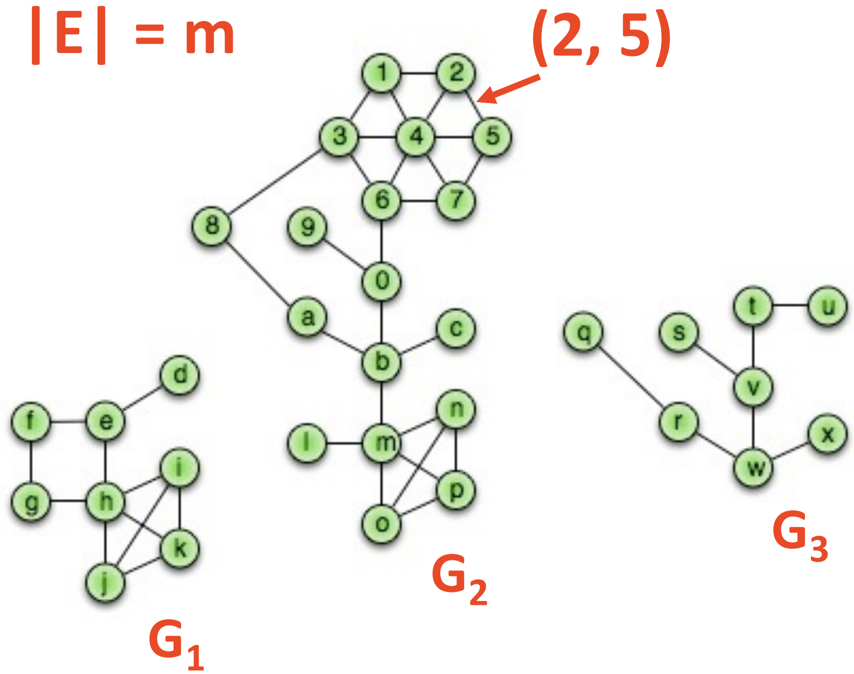
Simple Graph(G): A graph with no self loops or multi-edges.

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

$G' = (V', E')$:

$V' \in V$, $E' \in E$, and

$(u, v) \in E \rightarrow u \in V', v \in V'$

Complete subgraph(G)

Connected subgraph(G)

Connected component(G)

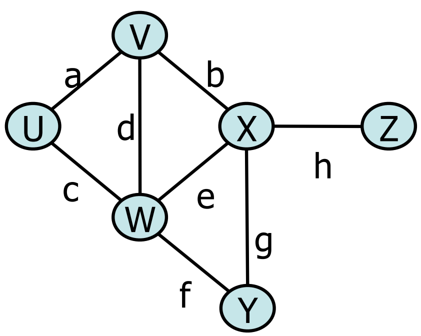
Acyclic subgraph(G)

Spanning tree(G)

Running times are often reported by **n**, the number of vertices, but often depend on **m**, the number of edges.

How many edges? **Minimum edges:**

Not Connected:



Connected*:

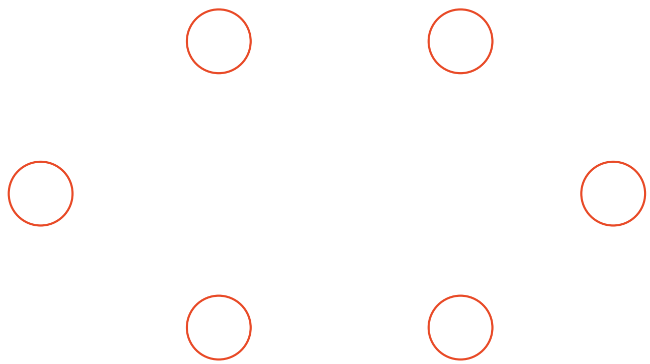
Maximum edges:

Simple:

Not simple:

$$\sum_{v \in V} \deg(v) =$$

Connected Graphs



Proving the size of a minimally connected graph

Theorem:
Every connected graph **G=(V, E)** has at least **|V|-1** edges.

Thm: Every connected graph **G=(V, E)** has at least **|V|-1** edges.

Proof: Consider an arbitrary, connected graph **G=(V, E)**.

Suppose |V| = 1:

Definition: A connected graph of 1 vertex has 0 edges.

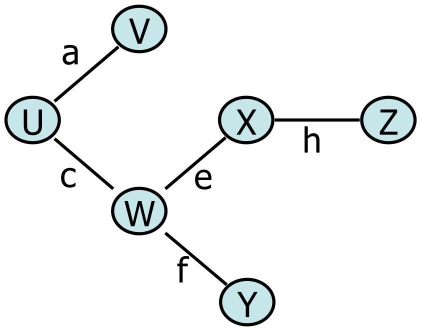
Theorem: |V|-1 edges → 1-1 = 0.

Inductive Hypothesis: For any **j < |V|**, any connected graph of **j** vertices has at least **j-1** edges.

Suppose |V| > 1:

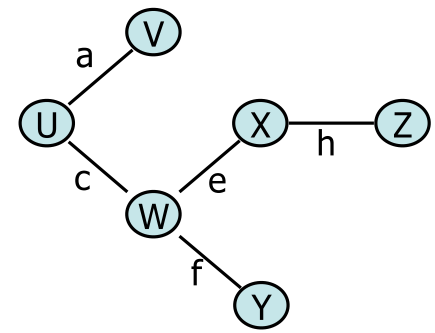
1. Choose any vertex:

2. Partition:



Suppose |V| > 1:

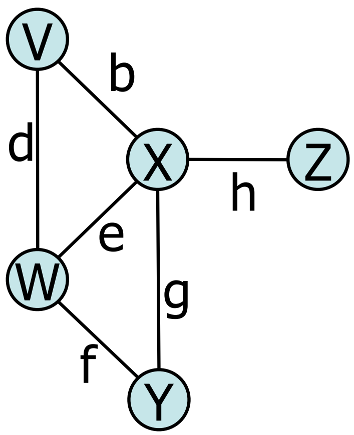
3. Count the edges



Graph ADT

Data:

- **Vertices**
- **Edges**
- **Some data structure maintaining the structure between vertices and edges.**



Functions:

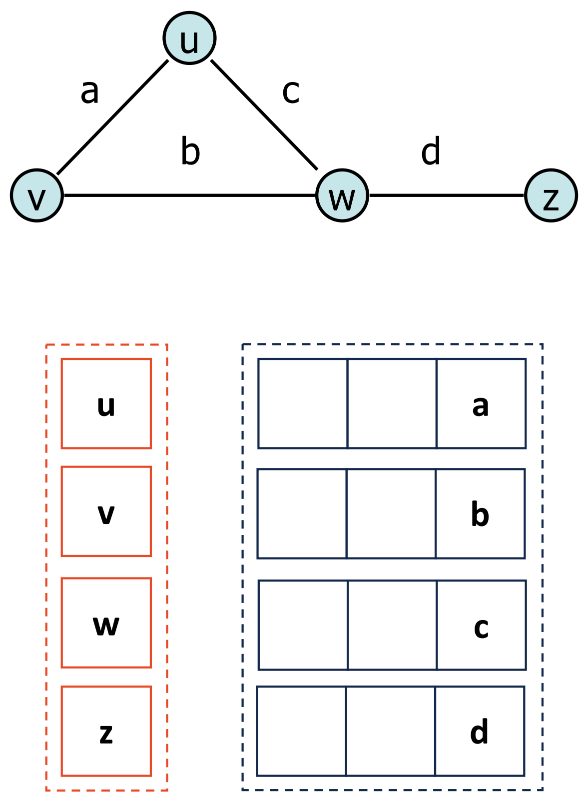
- **insertVertex(K key);**
- **insertEdge(Vertex v1, Vertex v2, K key);**

- **removeVertex(Vertex v);**
- **removeEdge(Vertex v1, Vertex v2);**

- **incidentEdges(Vertex v);**
- **areAdjacent(Vertex v1, Vertex v2);**

- **origin(Edge e);**
- **destination(Edge e);**

Graph Implementation: Edge List



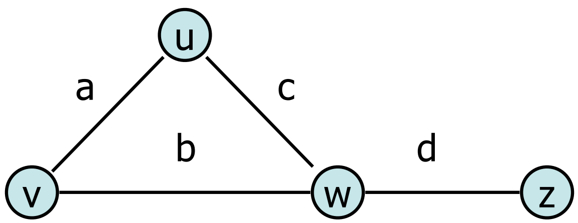
insertVertex(K key);

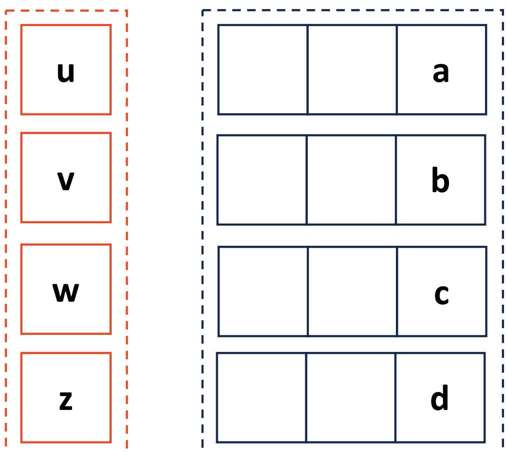
removeVertex(Vertex v);

areAdjacent(Vertex v1, Vertex v2);

incidentEdges(Vertex v);

Graph Implementation: Adjacency Matrix





insertVertex(K key);
removeVertex(Vertex v);
areAdjacent(Vertex v1, Vertex v2);
incidentEdges(Vertex v);

	u	v	w	z
u				
v				
w				
z				